PROGRAMACION ORIENTADA A OBJETOS

Temas de examen

27 DE ENERO DE 2023

ACADEMIA XIDERAL
Daniel de Jesus Molina Garcia

INDICE

Contenido

1 ¿Qué es SCRUM?

Scrum es un marco de trabajo para el desarrollo y mantenimiento de productos complejos o sencillos.

Es una de las metodologías agiles mas populares y usadas en proyectos de software, aunque una de sus ventajas es la adaptabilidad lo que la hace ideal para trabajar en diferentes contextos.

A grandes rasgos se compone de:

- Product Owner (dueño del producto).
- Scrum Master
- Development Team.

1. Product Owner (Dueño del producto):
   Es la representación del cliente dentro del equipo de trabajo, su principal responsabilidad es expresar claramente la necesidad del cliente dentro del Product Backlog.

2. Development Team:
   Los equipos deben ser pequeños, de tres a nueve personas por regla general. (pueden ser desarolladores, testers, analistas, entre otros)

3. Scrum Master:
   Ésta es la persona que capacitará al resto del equipo en el enfoque Scrum y que ayudará al equipo a eliminar todo lo que lo atrasa.

4. El Product Owner Define un documento que tiene una lista completa de funcionalidades con las necesidades del cliente (Product BackLog).
   En el BackLog van plasmados:
   - Necesidades
   - Ideas
   - Requisitos
   - Funcionalidades
   - Etc.

   Para cumplir la solicitud del cliente.

5. Afina y estima la bitácora del producto. Es crucial que la gente que realmente se hará cargo de los elementos de la bitácora del producto estime cuánto esfuerzo implicarán. El equipo debe examinar cada elemento de la bitácora y ver si, en efecto, es viable

6. Planeación del sprint. Ésta es la primera de las reuniones de Scrum. El equipo, el Scrum Master y el responsable del producto se sientan a planear el sprint. Los sprints son siempre de extensión fija, inferior a un mes.

7. Vuelve visible el trabajo. La forma más común de hacerlo en Scrum es crear una tabla de Scrum con tres columnas: Pendiente, En proceso y Terminado. Notas adhesivas representan los elementos por llevar a cabo y el equipo avanza por la tabla conforme los va concluyendo, uno por uno.

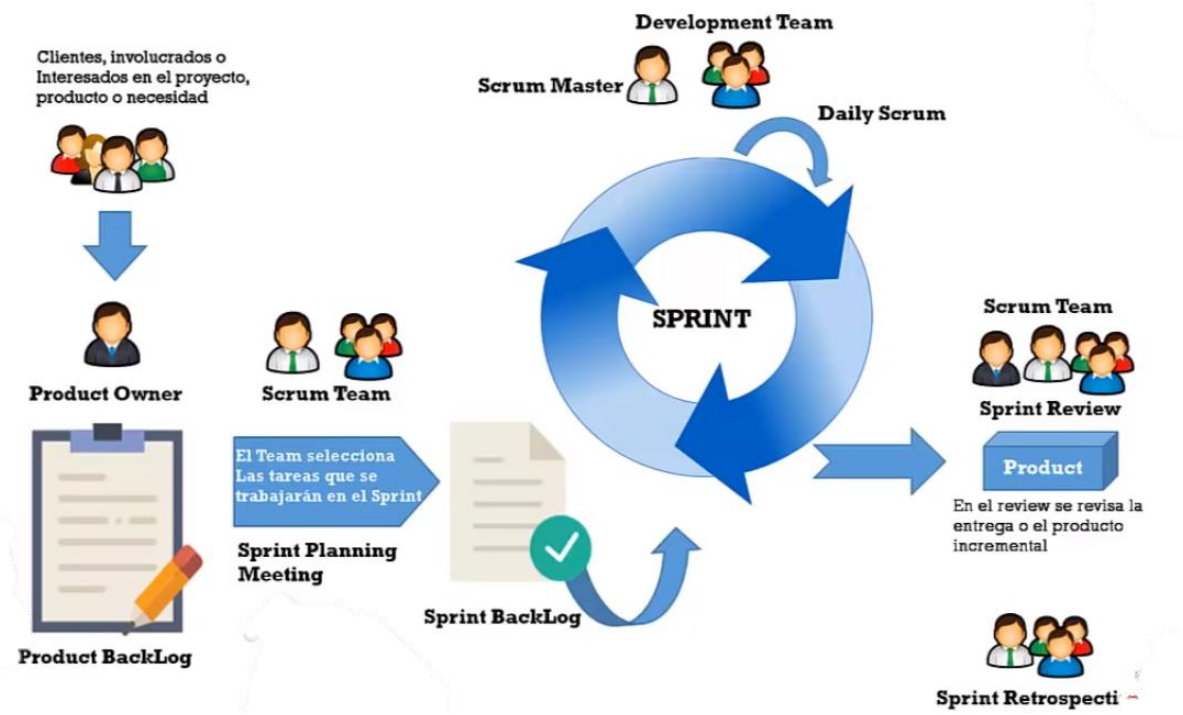
8. Daily Scrum. Éste es el pulso de Scrum. Cada día, a la misma hora, durante no más de quince minutos, el equipo y el Scrum Master se reúnen y contestan tres preguntas:

   ♦ ¿Qué hiciste ayer para ayudar al equipo a terminar el sprint?
   ♦ ¿Qué harás hoy para ayudar al equipo a terminar el sprint?
   ♦ ¿Algún obstáculo te impide o impide al equipo cumplir la meta del sprint?

9. Revisión del sprint o demostración del sprint. Ésta es la reunión en la que el equipo muestra lo que hizo durante el sprint. Todos pueden asistir, no sólo el responsable del producto, el Scrum Master y el equipo, sino también los demás interesados, la dirección, clientes, quien sea. Ésta es una reunión abierta en la que el equipo hace una demostración de lo que pudo llevar a Terminado durante el sprint.

10. Retrospectiva del sprint. Una vez que el equipo ha mostrado lo que logró en el sprint más reciente –la cosa “terminada” y en posibilidad de enviarse a los clientes en busca de realimentación–, piensa en qué marchó bien, qué pudo haber marchado mejor y qué puede mejorar en el siguiente sprint. ¿Cuál es la mejora en el proceso que como equipo pueden implementar de inmediato?

1.1 Ciclo de vida



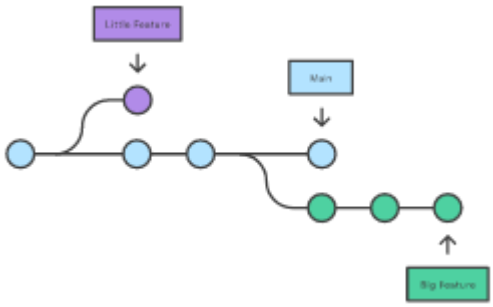
2 GitHub (branches, Merge y Conficts).

2.1 Git Branches

La creación de las ramas es una función en la mayoría de los sistemas de control de versiones modernos. En Git las ramas son parte del proceso para el desarrollo diario.

Los Branch son un puntero eficaz para añadir tus cambios o solucionar un error, independientemente de su tamaño generas una nueva rama para alojar estos cambios.

Esto hace que el código inestable se fusione con el código base principal y te da la oportunidad de limpiar tu historial antes de fusionarlo con la rama principal.



El comando git Branch te permite crear, enumerar y eliminar ramas, así como cambiar su nombre.

No te permite cambiar entre ramas o volver a unir in historial bifurcado. Por esto git Branch está estrechamente integrado con los comandos git checkout y git merge.

2.2 Git merge

Utilizamos este comando para combinar cambios de una rama a otra. Es decir la utilizamos para combinar los cambios de una rama especifica en la rama actual mediante la creación de un nuevo commit que contiene los cambios de ambas ramas.

Hay varios tipos de merge, los más comunes son:

Merge fast-forward: Este tipo de merge se produce cuando la rama actual no ha sido modificada desde el ultimo merge.

Merge con conflicto: Este tipo de merge es producido cuando ambas ramas han modificado el mismo archivo o línea. Git no puede determinar como combinarlos, por lo que se marca como conflicto y se lo deja al usuario que lo resuelva.

1.3 Conflictos

Varios desarrolladores pueden intentar editar el mismo contenido. Si el desarrollador A intenta editar el código que el desarrollador B esta editando, puede producir un conflicto.

Para disminuir la aparición de conflictos, los desarrolladores trabajaran en ramas separadas y aisladas.

3 Arquitectura MVC

MVC significa Modelo-Vista-Controlador. Es un patrón de diseño utilizado en la ingeniería de software para separar las preocupaciones de la gestión de datos (Modelo), interfaz de usuario (Ver) y control de flujo (Controlador) en un programa.

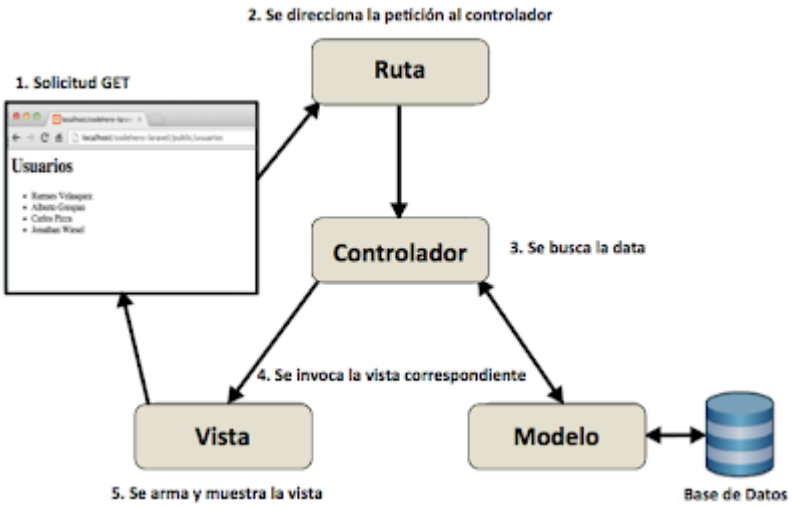
Un diagrama de MVC generalmente muestra tres componentes principales:

- Modelo
- Vista
- Controlador.

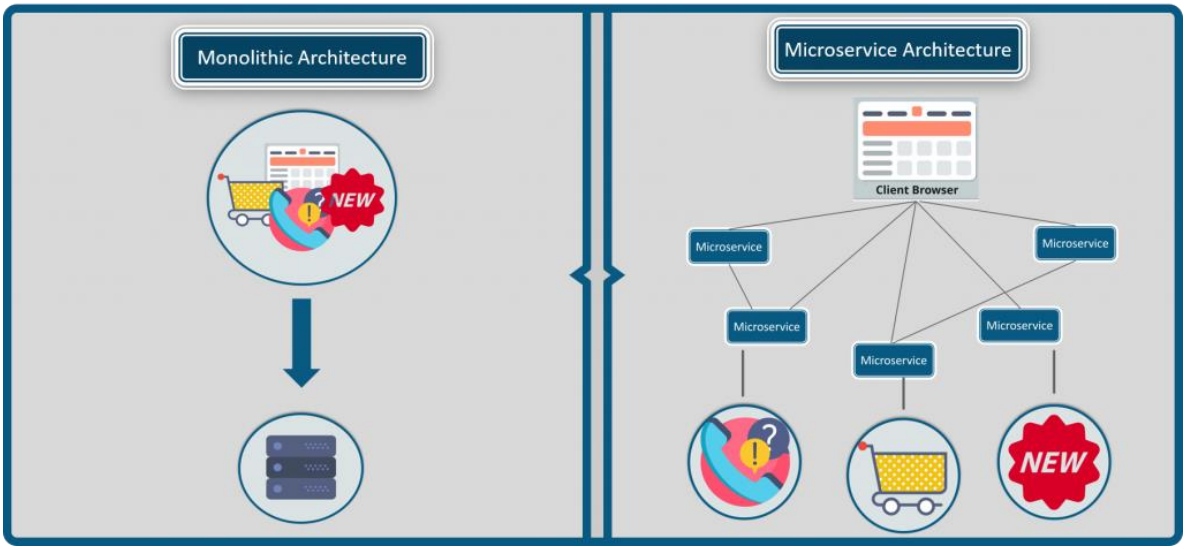
El modelo representa los datos y la lógica comercial de la aplicación, la vista es responsable de mostrar los datos al usuario, y el controlador maneja la comunicación entre el modelo y la vista.

El Modelo y la Vista a menudo se conectan a través de un patrón de observador, que permite que la Vista se actualice automáticamente cuando cambia el Modelo. El controlador recibe información del usuario y actualiza el modelo en consecuencia, y también actualiza la vista con los nuevos datos.

En resumen, MVC es un patrón de diseño que separa las preocupaciones de la gestión de datos, la interfaz de usuario y el flujo de control en un programa, y está representado por un diagrama donde se muestran tres componentes principales: Modelo, Vista y Controlador.



4 Arquitectura monolítica vs Arquitectura microservicios.



4.1 Arquitectura monolítica

es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. Para hacer cambios en este tipo de aplicación, hay actualizar toda la pila.

Ventajas:

Implementación sencilla: un único archivo o directorio ejecutable facilita la implementación.

Desarrollo: desarrollar una aplicación compilada con una única base de código es más sencillo.

Pruebas simplificadas:

Una aplicación monolítica es una unidad única y centralizada, por lo que las pruebas integrales se pueden hacer más rápido que con una aplicación distribuida.

Depuración sencilla: con todo el código ubicado en un solo lugar, es más fácil rastrear las solicitudes y localizar incidencias.

Desventajas:

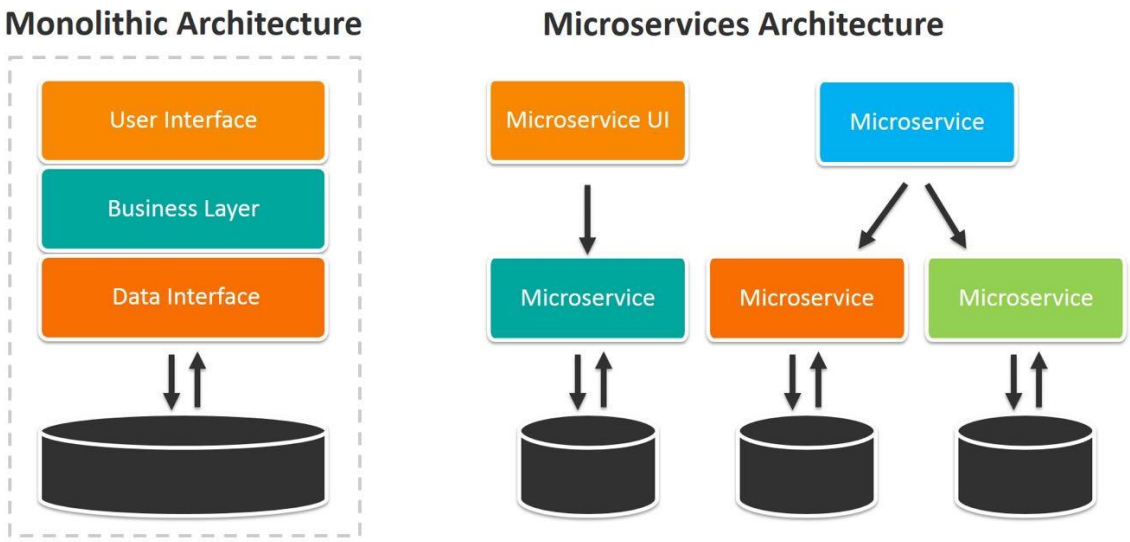
Velocidad de desarrollo más lenta: con una aplicación grande y monolítica, el desarrollo es más complejo y lento.

Escalabilidad: no se pueden escalar componentes individuales.

Fiabilidad: si hay un error en algún módulo, puede afectar a la disponibilidad de toda la aplicación.

Barrera para la adopción de tecnología: cualquier cambio en el marco o el lenguaje afecta a toda la aplicación, lo que hace que los cambios suelan ser costosos y lentos.

Implementación: un pequeño cambio en una aplicación monolítica requiere una nueva implementación de todo el monolito.



1.2 Microservicios:

Es un método de arquitectura que se basa en una serie de servicios que se pueden implementar de forma independiente. Estos servicios tienen su propia lógica empresarial y base de datos con un objetivo específico. La actualización, las pruebas, la implementación y el escalado se llevan a cabo dentro de cada servicio.

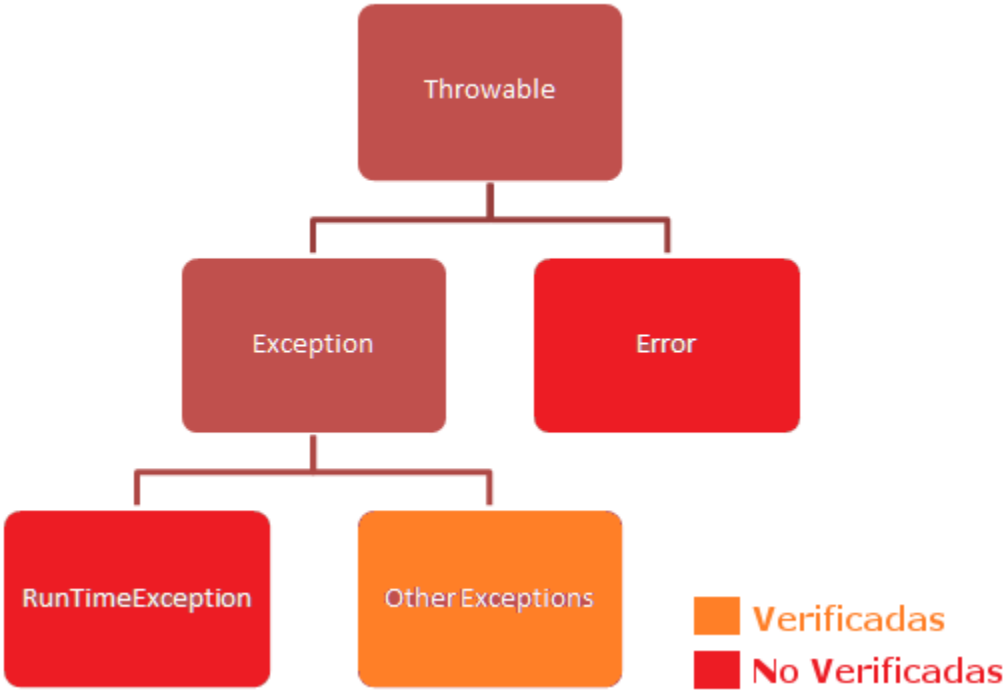
Ventajas:

Agilidad: promueve formas ágiles de trabajar con equipos pequeños que implementen con frecuencia.
Implementación continua: tenemos ciclos de lanzamiento frecuentes y más rápidos. Antes enviábamos actualizaciones una vez a la semana y ahora podemos hacerlo dos o tres veces al día.
Muy fácil de mantener y probar: los equipos pueden hacer pruebas con el código y dar marcha atrás si algo no funciona como esperan. Esto facilita la actualización del código y acelera el tiempo de salida al mercado de nuevas funciones.
Implementación independiente: los microservicios son unidades individuales, por lo que permiten una implementación independiente rápida y sencilla de funciones individuales.
Flexibilidad tecnológica: con las arquitecturas de microservicios, los equipos pueden elegir con libertad las herramientas que desean.
Alta fiabilidad: puedes implementar cambios para un servicio en concreto sin el riesgo de que se caiga toda la aplicación.

Desventajas:

Desarrollo descontrolado: los microservicios suman complejidad en comparación con las arquitecturas monolíticas, ya que hay más servicios en más lugares creados por varios equipos. Si el desarrollo descontrolado no se gestiona adecuadamente, se reduce la velocidad de desarrollo y el rendimiento operativo.
Costes exponenciales de infraestructura: cada nuevo microservicio puede tener su propio coste para el conjunto de pruebas, los manuales de estrategias de desarrollo, la infraestructura de alojamiento, las herramientas de supervisión, etc.
Más sobrecarga organizativa: los equipos deben agregar otro nivel de comunicación y colaboración para coordinar las actualizaciones e interfaces.
Desafíos para la depuración: cada microservicio tiene su propio conjunto de registros, lo que complica la depuración. Además, un solo proceso empresarial puede ejecutarse en varias máquinas, lo que aún lo dificulta más.
Falta de estandarización: sin una plataforma común, puede haber una proliferación de idiomas, de estándares de registro y de supervisión.
La propiedad no está clara: a medida que se introducen más servicios, también lo hace el número de equipos que los ejecutan. Con el tiempo, se hace difícil conocer los servicios disponibles que un equipo puede aprovechar y con quién hay que hablar para obtener asistencia.

5 Excepciones.



En java existen formas de manejar los errores, en el diagrama podemos observar cuales son de las que hablamos, a continuación, explicaremos estas excepciones.

Observamos que la clase Thowable hereda de la clase Object. Throwable es la clase que contiene a todas las excepciones que podemos utilizar a la hora de programar. Además de que podremos crear nuestras propias excepciones si así lo queremos, podemos heredarlo de la clase Exception o de alguna de sus siguientes sub-clases:

- Error:
  En este tipo de excepción lo arroja cuando el problema está en la JVM, son errores del sistema y no podremos hacer nada, solamente informar al usuario y cerrar el programa.
- Exception:
  En esta clase nos permitirá describir el tipo de error que entro en conflicto en el código, de esta manera el usuario no se le cerrará el programa de manera inesperada.
- RuntimeException:
  Esta excepción es lanzada durante el funcionamiento de la JVM Para ayudar a los programadores a anticipar y recuperarse de los errores de tiempo de ejecución.

6 Multicatch and TrywithResourse

El multicatch nos permite capturar varias excepciones diferentes en un solo bloque del catch, utilizando el carácter ”|” para separar cada excepción.

Por ejemplo:

```
Catch (IOException | SQLException e) {

}
```

El TryWithResourse nos permite cerrar recursos abiertos dentro de un bloque try automáticamente, de forma segura y evita fugas de recursos, dichos recursos tendrán que ser objetos que implementen de java.lang.AutoCloseable.

Por Ejemplo:

```
Try (FileInputStream input = new FileInputStream(“file.txt”)){

} catch (IOException e) { }
```

7 Tipos de collections.

7.1 List

Es una colección de elementos ordenados. Pueden ser de cualquier tipo de objeto, y pueden contener duplicados. La clase List es una interfaz, y existen varias implementaciones, como ArrayList, LinkedList, y Vector. ArrayList es la implementación más comúnmente utilizada, ya que es rápida y fácil de usar. LinkedList es útil para operaciones de inserción y eliminación en cualquier lugar de la lista, mientras que Vector es una implementación antigua de List que es sincronizada y se utiliza menos a menudo.

7.2 Queue

Es una colección de elementos ordenados según su orden de llegada. Es similar a una cola en la vida real, donde los elementos entran por un extremo (el "final" de la cola) y salen por el otro extremo (el "frente" de la cola). Los métodos más comunes en una cola son offer, poll, peek.

7.3 Set

Es una colección de elementos únicos, es decir, no permite elementos duplicados. A diferencia de List, no mantiene un orden específico entre los elementos. A menudo

se utiliza para verificar si un elemento ya ha sido agregado a un conjunto o para eliminar duplicados de una lista.

La interfaz Set es una colección de elementos no ordenados y no indexados, y existen varias implementaciones, como HashSet, LinkedHashSet, y TreeSet.

7.4 Map

Es una colección de pares clave-valor. Cada elemento en una Map tiene una clave única asociada a él, y se utiliza para acceder al valor del elemento. Es similar a un diccionario en la vida real, donde las palabras son las claves y las definiciones son los valores.

La interfaz Map es una colección no ordenada y no indexada, y existen varias implementaciones, como HashMap, LinkedHashMap, y TreeMap.